

Bölüm 4: Değişkenler

Mikroişlemciler

DB / DW / DD

Adlandırma kuralları

Bellek yerleşimi

ORG 100h ve .COM

Diziler ve DUP

LEA ve OFFSET

EQU sabitleri

Toplama ve karşılaştırma

Çift sözcük

Sercan KÜLCÜ · Giresun Üniversitesi · Elektrik-Elektronik Mühendisliği

Öğrenme Çıktıları

Tanımlama

DB, DW, DD, DUP ve ? ile değişken ve dizi tanımlamak.

Bellek yerleşimi

Derleyicinin değişkenleri hangi bağlı konumlara, hangi bayt sırasıyla yerleştirdiğini çıkarmak.

Adresler

LEA ve OFFSET ile değişken adresini almak ve dizi elemanlarına indisle erişmek.

Programlar

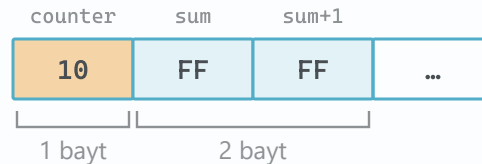
Değişkenler arasında veri taşıyan, toplayan ve karşılaştıran programları adım adım izlemek.

DEĞİŞKENLER

Değişkenler

- Bir değişken bir **bellek konumunu** temsil eder.
- Programcı için `var1` adını kullanmak, `5A73:235B` adresini kullanmaktan çok daha kolaydır.
- Derleyici iki temel tür destekler:
 - **BYTE** (8 bit): `ad DB değer` (Define Byte)
 - **WORD** (16 bit): `ad DW değer` (Define Word)
- `DD` (Define Double word, 32 bit) de vardır; bölüm sonunda ele alınır.

```
counter DB 10h      ; BYTE  
sum      DW 0FFFFh   ; WORD
```



DEĞİŞKENLER

Adlandırma ve İlk Değer

- Değişken adları **harfle başlamalıdır**; sonrasında harf ve rakam kombinasyonları içerebilir.
- Yazmaç ve komut adları (`AX` , `MOV` ...) değişken adı olamaz.
- Ad belirtilmeyen değişkenler de olabilir: adları yoktur ama **adresleri** vardır.
- Herhangi bir sayısal değer (ikili, onluk, onaltılık, karakter) alabilir.
- İlk değer atanmamış değişkenler için `?` kullanılır (emu8086 belleğe 00 koyar).

Tanım	?
<code>sayi1 DB 5</code>	geçerli
<code>1sayi DB 5</code>	rakamla başlıyor
<code>ax DW 0</code>	yazmaç adı
<code>DB 0Ah</code>	adsız, ama adresi var
<code>sonuc DW ?</code>	ilk değersiz
<code>harf DB 'A'</code>	41h

DEĞİŞKENLER

Değer Aralıkları

Direktif	Boyut	İşaretsiz	İşaretli	Örnek
DB	1 bayt	0 ... 255	-128 ... 127	x DB -1 → FFh
DW	2 bayt	0 ... 65535	-32768 ... 32767	y DW -2 → FE FF
DD	4 bayt	0 ... 4 294 967 295	$-2^{31} \dots 2^{31}-1$	z DD 1 → 01 00 00 00

Bellekte yalnızca bit deseni saklanır; **işaretli mi işaretsiz mi** olduğu, değeri kullanan komuta bağlıdır (ör. JGE işaretli, JAE işaretsiz karşılaştırır). DB 300 bir bayta sığmaz → assembler hata verir.

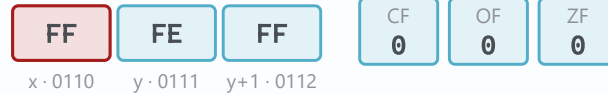
Kod İzleme: FFh 255 mi, -1 mi?

```

1  ORG 100h
2  ► MOV AL, x      ; FFh
3  ADD AL, 1        ; 255 + 1 ?
4  MOV AL, x
5  CBW              ; işaretli: -1
6  MOV BX, y        ; FFEh
7  NEG BX           ; -(-2)
8  RET
9  x DB -1
10 y DW -2

```

bellek



AX **00FFh** işaretli 255 AL: 255 / -1
 BX **0000h** işaretli 0

Bellekte x = **FFh**, y = **FE FF** (little-endian FFEh). AL = x = FFh: işaretli 255, işaretli -1.

Adım 1 / 6

DEĞİŞKENLER

AX = **00FFh**BX = **0000h**CF = **0**ZF = **0**

DB/DW yalnızca bit desenini saklar; işaretli mi işaretli mi olduğuna değeri kullanan komut karar verir.

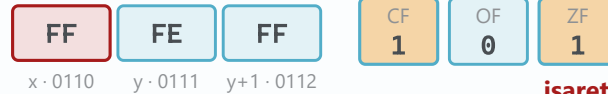
Kod İzleme: FFh 255 mi, -1 mi?

```

1  ORG 100h
2  MOV AL, x      ; FFh
3  ADD AL, 1      ; 255 + 1 ?
4  MOV AL, x
5  CBW           ; işaretli: -1
6  MOV BX, y      ; FFEh
7  NEG BX         ; -(-2)
8  RET
9  x DB -1
10 y DW -2

```

bellek

işaretsiz taşıma: **CF**

AX **0000h** işaretsiz 0
işaretli 0

AL: 0 / 0

BX **0000h** işaretsiz 0
işaretli 0

ADD AL, 1 : işaretsiz bakışla $255 + 1 = 256$ bayta sığmaz →
AL = **00h**, **CF** = **1**. İşaretli bakışla $-1 + 1 = 0$ doğru: **OF** = 0.

Adım 2 / 6

DEĞİŞKENLER

AX = **0000h**BX = **0000h**CF = **1**ZF = **1**

DB/DW yalnızca bit desenini saklar; işaretli mi işaretsiz mi olduğuna değeri kullanan komut karar verir.

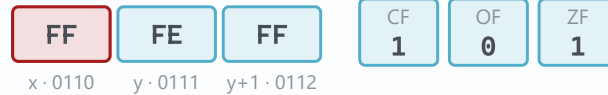
Kod İzleme: FFh 255 mi, -1 mi?

```

1  ORG 100h
2  MOV AL, x      ; FFh
3  ADD AL, 1      ; 255 + 1 ?
4  MOV AL, x
5  CBW           ; işaretli: -1
6  MOV BX, y      ; FFEh
7  NEG BX        ; -(-2)
8  RET
9  x DB -1
10 y DW -2

```

bellek



AX

00FFh

işaretsiz 255
işaretli 255

AL: 255 / -1

BX

0000h

işaretsiz 0
işaretli 0

AL yeniden FFh.

Adım 3 / 6

DEĞİŞKENLER

AX = **00FFh**BX = **0000h**CF = **1**ZF = **1**

DB/DW yalnızca bit desenini saklar; işaretli mi işaretsiz mi olduğuna değeri kullanan komut karar verir.

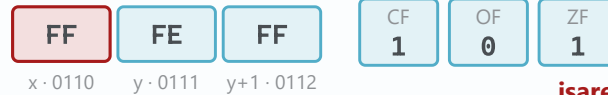
Kod İzleme: FFh 255 mi, -1 mi?

```

1  ORG 100h
2  MOV AL, x      ; FFh
3  ADD AL, 1      ; 255 + 1 ?
4  MOV AL, x
5  CBW           ; işaretli: -1
6  MOV BX, y      ; FFEh
7  NEG BX        ; -(-2)
8  RET
9  x DB -1
10 y DW -2

```

bellek



işaret genişletme

AX **FFFFh** işaretli 65535
işaretsiz 65535
işaretli -1 AL: 255 / -1

BX **0000h** işaretli 0
işaretsiz 0
işaretli 0

CBW AL'yi işaretli kabul eder: MSB = 1 → AH = FFh, AX = **FFFFh** = -1 (16 bit).

Adım 4 / 6

DEĞİŞKENLER

AX = **FFFFh**BX = **0000h**CF = **1**ZF = **1**

DB/DW yalnızca bit desenini saklar; işaretli mi işaretsiz mi olduğuna değeri kullanan komut karar verir.

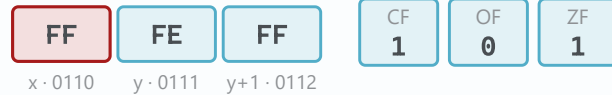
Kod İzleme: FFh 255 mi, -1 mi?

```

1  ORG 100h
2  MOV AL, x      ; FFh
3  ADD AL, 1      ; 255 + 1 ?
4  MOV AL, x
5  CBW           ; işaretli: -1
6  MOV BX, y      ; FFEh
7  NEG BX         ; -(-2)
8  RET
9  x DB -1
10 y DW -2

```

bellek



AX **FFFFh** işaretli 65535
işaretsiz 65535 AL: 255 / -1

BX **FFFEh** işaretli 65534
işaretsiz 65534 işaretli -2

BX = y = **FFFEh**: işaretli 65534, işaretli -2.

Adım 5 / 6

DEĞİŞKENLER

AX = **FFFFh**BX = **FFFEh**CF = **1**ZF = **1**

DB/DW yalnızca bit desenini saklar; işaretli mi işaretsiz mi olduğuna değeri kullanan komut karar verir.

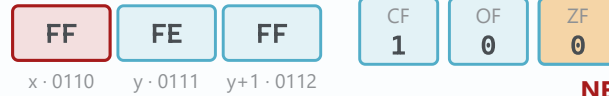
Kod İzleme: FFh 255 mi, -1 mi?

```

1  ORG 100h
2  MOV AL, x      ; FFh
3  ADD AL, 1      ; 255 + 1 ?
4  MOV AL, x
5  CBW           ; işaretli: -1
6  MOV BX, y      ; FFFEh
7  NEG BX         ; -(-2)
8  RET
9  x DB -1
10 y DW -2

```

bellek



AX **FFFFh** işaretli 65535
işaretsiz -1 AL: 255 / -1

BX **0002h** işaretli 2
işaretsiz 2

NEG BX : $0 - (-2) = \mathbf{0002h}$. Bit deseninin anlamını komut seçti: ADD/CF işaretli, CBW/NEG işaretli yorumladı.

Adım 6 / 6

DEĞİŞKENLER

AX = **FFFFh**BX = **0002h**CF = **1**ZF = **0**

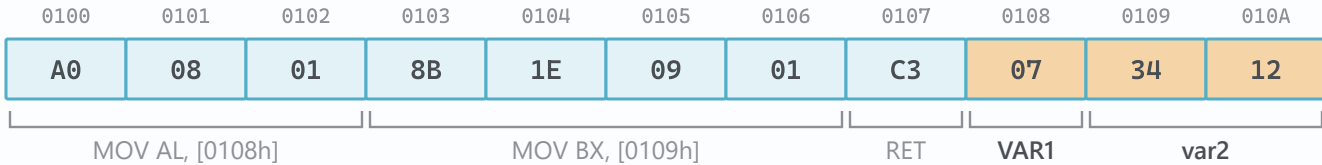
DB/DW yalnızca bit desenini saklar; işaretli mi işaretsiz mi olduğuna değeri kullanan komut karar verir.

BELLEK YERLEŞİMİ

Örnek Kod

```
ORG 100h
MOV AL, var1 ; AL ← var1
MOV BX, var2 ; BX ← var2
RET          ; programı bitir
VAR1 DB 7    ; BYTE, 7
var2 DW 1234h ; WORD, 1234h
```

- Kod önce, değişkenler **RET'ten sonra**: CPU onlara hiç ulaşmaz.
- Derleyici kaynağı bir **bayt dizisine** çevirir ve değişken adlarını bellek konumlarıyla değiştirir.
- Büyük-küçük harfe duyarlıdır: **VAR1** ile **var1** aynı değişkendir.



Kod İzleme: var1 ve var2

► **ORG** 100h

```
2 MOV AL, var1 ; AL ← var1
3 MOV BX, var2 ; BX ← var2
4 RET          ; programı bitir
5 VAR1 DB 7    ; BYTE, 7
6 var2 DW 1234h ; WORD, 1234h
```

AX

00	00
AH	AL

 BX

00	00
BH	BL

 IP

0100

0B56:0107 ... 010A

0107	0108	0109	010A
C3	07	34	12

Program 0B56:0100'e yüklendi; DS = CS = 0B56h, IP = 0100h.

Adım 1 / 4

DEĞİŞKENLER

IP = 0100h

AL = 00h

BX = 0000h

Kod İzleme: var1 ve var2

1 **ORG** 100h

► **MOV** AL, var1 ; AL ← var1

3 **MOV** BX, var2 ; BX ← var2

4 **RET** ; programı bitir

5 VAR1 **DB** 7 ; BYTE, 7

6 var2 **DW** 1234h ; WORD, 1234h

AX

00

07

AH

AL

BX

00

00

BH

BL

IP

0103

0B56:0107 ... 010A

0107

0108

0109

010A

C3

07

34

12

MOV AL, [0108h] : VAR1'in değeri → AL = **07h**. IP = 0103h.

Adım 2 / 4

DEĞİŞKENLER

IP = 0103h

AL = 07h

BX = 0000h

Kod İzleme: var1 ve var2

```
1 ORG 100h
2 MOV AL, var1 ; AL ← var1
3 MOV BX, var2 ; BX ← var2
4 RET ; programı bitir
5 VAR1 DB 7 ; BYTE, 7
6 var2 DW 1234h ; WORD, 1234h
```

AX 00 07 BX 12 34 IP 0107
AH AL BH BL

0B56:0107 ... 010A

0107 0108 0109 010A
C3 07 34 12

MOV BX, [0109h] : 34h düşük, 12h yüksek bayt → BX = **1234h**. IP = 0107h.

Adım 3 / 4

DEĞİŞKENLER

IP = **0107h**

AL = **07h**

BX = **1234h**

Kod İzleme: var1 ve var2

```
1 ORG 100h
2 MOV AL, var1 ; AL ← var1
3 MOV BX, var2 ; BX ← var2
4 RET ; programı bitir
5 VAR1 DB 7 ; BYTE, 7
6 var2 DW 1234h ; WORD, 1234h
```

AX **00** **07** BX **12** **34** IP **0107**
AH AL BH BL

0B56:0107 ... 010A

0107 0108 0109 010A
C3 **07** **34** **12**

RET (0107h'de C3) programı bitirir. 0108h'den sonrası veridir, komut olarak yürütülmez.

Adım 4 / 4

DEĞİŞKENLER

IP = **0107h**

AL = **07h**

BX = **1234h**

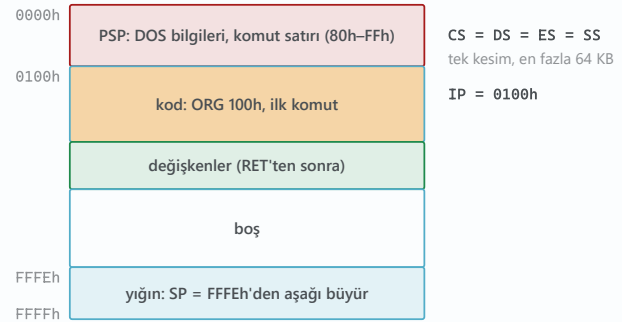
Derleyici Ne Yapar?

- Kaynak kodu bir dizi bayta dönüştürür; değişken adlarını **bağıl konumlarla** değiştirir:
`MOV AL, var1` → `MOV AL, [0108h]` .
- .COM dosyaları yüklendiğinde **DS, CS yazmacının değerini alır** (ES ve SS de aynıdır).
- **Bağıl konum** (offset), bellek kesiminin başlangıcından itibaren olan uzaklıktır.
- `VAR1` 'in bağıl konumu **0108h**, tam adresi **0B56:0108**'dir.
- `var2` 'nin bağıl konumu **0109h**, tam adresi **0B56:0109**'dur.
 - WORD türünde olduğu için 2 bayt kaplar (0109h ve 010Ah).
 - Düşük bayt düşük adreste saklanır: **34h, 12h'den önce** yer alır (little-endian).

Kesim değeri (0B56h) DOS'un programı nereye yüklediğine bağlıdır ve her çalıştırmada farklı olabilir; bağıl konumlar ise derleme anında sabittir.

ORG 100h Direktifi

- Derleyiciye çalıştırılabilir dosyanın **yükleneceği adresi** söyler; değişken adları adreslerle değiştirilirken dikkate alınır.
- **.COM** dosyalarında kullanılır.
- İşletim sistemi kesimin **ilk 256 baytında** (PSP) programla ilgili verileri tutar; komut satırı parametreleri bu alandadır.
- **.EXE** dosyaları 0000h bağlı konumuna yüklenir ve değişkenler için ayrı bir kesim kullanır.
- Direktifler gerçek makine koduna **dönüştürülmez**.



BELLEK YERLEŞİMİ

.COM ve .EXE

	.COM	.EXE
Başlangıç	<code>ORG 100h</code> , IP = 0100h	giriş noktası <code>END start</code> ile belirtilir
Kesimler	tek kesim: CS = DS = ES = SS	ayrı kod, veri ve yığın kesimleri
Boyut	en fazla 64 KB (PSP dahil)	64 KB'tan büyük olabilir
DS	hazır, değişkenlere doğrudan erişilir	programın başında <code>MOV AX, @data</code> · <code>MOV DS, AX</code>
Dosya	yalnızca makine kodu (başlık yok)	yeniden yerleştirme (relocation) bilgili başlık

.COM: tek kesim (64 KB)



.EXE: ayrı kesimler



Bu dersteki örneklerin çoğu emu8086 .COM şablonuyla yazılmıştır.

DİZİLER

Diziler

- Diziler **değişken zincirleri** olarak düşünülebilir: ardışık bellek konumlarında saklanan veri gruplarıdır.
- Tırnak içindeki metin otomatik olarak **bayt dizisine** (ASCII kodları) dönüştürülür.
- Dizi elemanlarına bellek adresi üzerinden erişilir: ilk eleman dizinin adresinde, sonrakiler ardından.

a **DB** 48h, 65h, 6Ch, 6Ch, 6Fh, 00h

b **DB** 'Hello', 0

...	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	b[0]	b[1]	b[2]	
	48	65	6C	6C	6F	00	48	65	6C	...

a ve b bellekte aynı baytları tutar: 48h 'H', 65h 'e', 6Ch 'l', 6Ch 'l', 6Fh 'o', 00h (dizgi sonu).

Kod İzleme: Dizi Elemanlarına Erişim

```
► MOV AL, a[3] ; a + 3
2 MOV SI, 0
3 MOV BL, a[SI] ; a + 0
4 INC SI
5 MOV BH, a[SI] ; a + 1
6 MOV a[SI], 'E' ; a[1] ← 45h
7 RET
8 a DB 48h,65h,6Ch,6Ch,6Fh,0
```

AX

00	00
AH	AL

 BX

00	00
BH	BL

 SI

0000

a dizisi

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]
48	65	6C	6C	6F	00

a[3], **a + 3** bayt demektir: AL = **6Ch** ('l').

Adım 1 / 7

DEĞİŞKENLER

AL = 00h

SI = 0000h

BL = 00h

BH = 00h

Kod İzleme: Dizi Elemanlarına Erişim

```
1 MOV AL, a[3] ; a + 3
▶ MOV SI, 0
3 MOV BL, a[SI] ; a + 0
4 INC SI
5 MOV BH, a[SI] ; a + 1
6 MOV a[SI], 'E' ; a[1] ← 45h
7 RET
8 a DB 48h,65h,6Ch,6Ch,6Fh,0
```

AX

00	00
AH	AL

 BX

00	00
BH	BL

 SI

0000

a dizisi

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]
48	65	6C	6C	6F	00

İndis yazmacı SI = 0.

Adım 2 / 7

DEĞİŞKENLER

AL = 00h

SI = 0000h

BL = 00h

BH = 00h

Kod İzleme: Dizi Elemanlarına Erişim

```
1 MOV AL, a[3] ; a + 3
2 MOV SI, 0
▶ 3 MOV BL, a[SI] ; a + 0
4 INC SI
5 MOV BH, a[SI] ; a + 1
6 MOV a[SI], 'E' ; a[1] ← 45h
7 RET
8 a DB 48h,65h,6Ch,6Ch,6Fh,0
```

AX 00 00 BX 00 48 SI 0000
AH AL BH BL

a dizisi

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]
48	65	6C	6C	6F	00

a[SI] = [a + SI]: BL = **48h** ('H').

Adım 3 / 7

DEĞİŞKENLER

AL = 00h

SI = 0000h

BL = 48h

BH = 00h

Kod İzleme: Dizi Elemanlarına Erişim

```
1 MOV AL, a[3] ; a + 3
2 MOV SI, 0
3 MOV BL, a[SI] ; a + 0
▶ INC SI
5 MOV BH, a[SI] ; a + 1
6 MOV a[SI], 'E' ; a[1] ← 45h
7 RET
8 a DB 48h,65h,6Ch,6Ch,6Fh,0
```

AX 00 00 BX 00 48 SI 0001
AH AL BH BL

a dizisi

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]
48	65	6C	6C	6F	00

SI = 1: bir sonraki bayt.

Adım 4 / 7

DEĞİŞKENLER

AL = 00h

SI = 0001h

BL = 48h

BH = 00h

Kod İzleme: Dizi Elemanlarına Erişim

```
1 MOV AL, a[3] ; a + 3
2 MOV SI, 0
3 MOV BL, a[SI] ; a + 0
4 INC SI
▶ 5 MOV BH, a[SI] ; a + 1
6 MOV a[SI], 'E' ; a[1] ← 45h
7 RET
8 a DB 48h,65h,6Ch,6Ch,6Fh,0
```

AX 00 00 BX 65 48 SI 0001
AH AL BH BL

a dizisi

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]
48	65	6C	6C	6F	00

BH = a[1] = **65h** ('e'). BX = 6548h.

Adım 5 / 7

DEĞİŞKENLER

AL = 00h

SI = 0001h

BL = 48h

BH = 65h

Kod İzleme: Dizi Elemanlarına Erişim

```
1 MOV AL, a[3] ; a + 3
2 MOV SI, 0
3 MOV BL, a[SI] ; a + 0
4 INC SI
5 MOV BH, a[SI] ; a + 1
▶ 6 MOV a[SI], 'E' ; a[1] ← 45h
7 RET
8 a DB 48h,65h,6Ch,6Ch,6Fh,0
```

AX 00 00 BX 65 48 SI 0001
AH AL BH BL

a dizisi

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]
48	45	6C	6C	6F	00

Dizi bayt türünde olduğu için boyut belli: a[1] ← **45h** ('E'). Dizi artık "Hello".

Adım 6 / 7

DEĞİŞKENLER

AL = 00h

SI = 0001h

BL = 48h

BH = 65h

Kod İzleme: Dizi Elemanlarına Erişim

```
1 MOV AL, a[3] ; a + 3
2 MOV SI, 0
3 MOV BL, a[SI] ; a + 0
4 INC SI
5 MOV BH, a[SI] ; a + 1
6 MOV a[SI], 'E' ; a[1] ← 45h
▶ RET
8 a DB 48h,65h,6Ch,6Ch,6Fh,0
```

AX 00 00 BX 65 48 SI 0001
AH AL BH BL

a dizisi

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]
48	45	6C	6C	6F	00

RET. Dizi tanımı kodun sonunda, yürütülmeyen bölgede.

Adım 7 / 7

DEĞİŞKENLER

AL = 00h

SI = 0001h

BL = 48h

BH = 65h

DİZİLER

Erişim Yolları ve DUP

Elemana erişim

- Köşeli parantezle sabit indis: `MOV AL, a[3]`
- Bellek indis yazmaçları (BX, SI, DI, BP) ile:
`MOV SI, 3` · `MOV AL, a[SI]`
- Eşdeğer yazımlar: `a[SI] = [a + SI] = [SI + a]`
- İndis **bayt** cinsindendir, eleman sayısı değil.

DUP işleci

Büyük dizi için: `sayı DUP(değer(ler))`

- `c DB 5 DUP(9)` = `c DB 9, 9, 9, 9, 9`
- `d DB 5 DUP(1, 2)` = `d DB 1, 2, 1, 2, 1, 2, 1, 2, 1, 2`
- `tampon DB 100 DUP(?)` : 100 baytlık ilk değersiz alan
- `tablo DW 10 DUP(0)` : 10 sözcük = 20 bayt

`c DB 5 DUP(9)`

09	09	09	09	09
----	----	----	----	----

 5 bayt

`d DB 5 DUP(1, 2)`

01	02	01	02	01	02	01	02	01	02
----	----	----	----	----	----	----	----	----	----

 10 bayt

DW ile **sayı dizileri** tanımlanabilir (`w DW 1000h, 2000h`); kullanılmadığı şey **metin dizileridir** (`DW 'Hello'` geçersiz). Değerler 255'ten büyük ya da -128'den küçükse DW gerekir.

Kod İzleme: DW Dizisinde İndis

```
► MOV AX, w[0] ; 1. eleman
2 MOV BX, w[2] ; 2. eleman
3 MOV SI, 4
4 MOV CX, w[SI] ; 3. eleman
5 MOV DX, w[1] ; dikkat!
6 w DW 1111h, 2222h, 3333h
```



w dizisi (bayt bayt)



w[0]: ilk sözcük → AX = **1111h**.

Adım 1 / 5

DEĞİŞKENLER

AX = 0000h

BX = 0000h

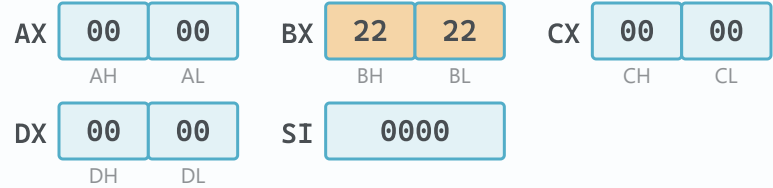
SI = 0000h

CX = 0000h

DX = 0000h

Kod İzleme: DW Dizisinde İndis

```
1 MOV AX, w[0] ; 1. eleman
▶ MOV BX, w[2] ; 2. eleman
3 MOV SI, 4
4 MOV CX, w[SI] ; 3. eleman
5 MOV DX, w[1] ; dikkat!
6 w DW 1111h, 2222h, 3333h
```



w dizisi (bayt bayt)



İkinci eleman **w[2]**'dedir (her eleman 2 bayt): BX = 2222h.

Adım 2 / 5

DEĞİŞKENLER

AX = 0000h

BX = 2222h

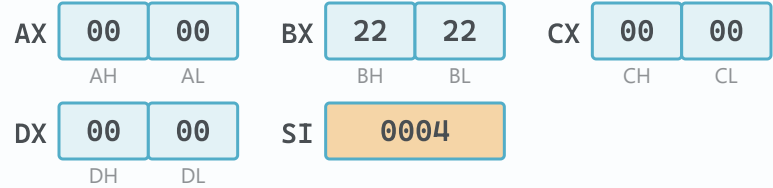
SI = 0000h

CX = 0000h

DX = 0000h

Kod İzleme: DW Dizisinde İndis

```
1 MOV AX, w[0] ; 1. eleman
2 MOV BX, w[2] ; 2. eleman
▶ MOV SI, 4
4 MOV CX, w[SI] ; 3. eleman
5 MOV DX, w[1] ; dikkat!
6 w DW 1111h, 2222h, 3333h
```



w dizisi (bayt bayt)

w+0	w+1	w+2	w+3	w+4	w+5
11	11	22	22	33	33

i. elemanın indisi = $2 \times i$: üçüncü eleman için SI = 4.

Adım 3 / 5

DEĞİŞKENLER

AX = 0000h

BX = 2222h

SI = 0004h

CX = 0000h

DX = 0000h

Kod İzleme: DW Dizisinde İndis

```
1 MOV AX, w[0] ; 1. eleman
2 MOV BX, w[2] ; 2. eleman
3 MOV SI, 4
▶ MOV CX, w[SI] ; 3. eleman
5 MOV DX, w[1] ; dikkat!
6 w DW 1111h, 2222h, 3333h
```



w dizisi (bayt bayt)

w+0	w+1	w+2	w+3	w+4	w+5
11	11	22	22	33	33

CX = w[SI] = **3333h**.

Adım 4 / 5

DEĞİŞKENLER

AX = 0000h

BX = 2222h

SI = 0004h

CX = 3333h

DX = 0000h

Kod İzleme: DW Dizisinde İndis

```
1 MOV AX, w[0] ; 1. eleman
2 MOV BX, w[2] ; 2. eleman
3 MOV SI, 4
4 MOV CX, w[SI] ; 3. eleman
▶ MOV DX, w[1] ; dikkat!
6 w DW 1111h, 2222h, 3333h
```



w dizisi (bayt bayt)



w[1] elemanın ortasından okur: baytlar 11h, 22h → DX = **2211h**. Sık yapılan hata!

Adım 5 / 5

DEĞİŞKENLER

AX = 0000h

BX = 2222h

SI = 0004h

CX = 3333h

DX = 2211h

Tanımdan Bayta

```
b DB 'Hello', 0
```

DB liste

DW

DUP

DW ?, -1

DD

hata

+0	+1	+2	+3	+4	+5
48	65	6C	6C	6F	00

b: 6 bayt · öge boyutu 1 bayt · renk değişimi = bir öge

Bir DB / DW / DD satırı yazın: sayılar (onluk, **h**, **b**, **o**), **'metin'** (yalnız DB), **?** ve **n DUP(...)** desteklenir. Hücre üstü: değişkenin başından bayt uzaklığı.

ADRESLER

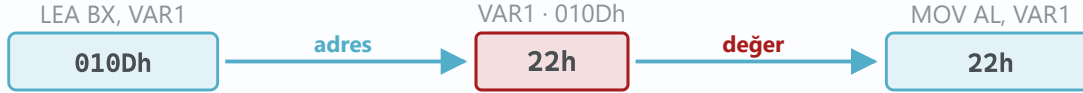
Değişkenin Adresini Alma

LEA (Load Effective Address)

Değişkenin **adresini** (bağlı konum) yazmaca yükler: `LEA BX, VAR1`. İndisli değişkenlerin adresini almak için de kullanılabilir: `LEA BX, a[SI]`.

OFFSET

Assembler işlecidir; değişkenin bağlı konum **değerini** derleme anında sağlar: `MOV BX, OFFSET VAR1`.



Değişkenin adresi, bir **prosedüre parametre geçirmek** için çok kullanışlıdır: prosedüre büyük bir dizinin kendisi değil, yalnızca 16 bitlik adresi verilir.

Komut	BX'e ne yüklenir?
<code>MOV BX, VAR1</code> (VAR1 DW)	VAR1'in değeri (içerik)
<code>LEA BX, VAR1</code> · <code>MOV BX, OFFSET VAR1</code>	VAR1'in adres i (bağlı konum)

Kod İzleme: LEA ile Değişken Güncelleme

► **ORG** 100h

```
2 MOV AL, VAR1          ; değer
3 LEA BX, VAR1          ; adres
4 MOV BYTE PTR [BX], 44h
5 MOV AL, VAR1          ; yeni değer
6 RET
7 VAR1 DB 22h
8 END
```

AX	00	00	BX	00	00
	AH	AL		BH	BL

DS:010C ... (VAR1 = 010Dh)

010C	010D	010E
C3	22	00

Kod 0100h'den başlar; derleyici VAR1'i **010Dh**'ye yerleştirdi (12 bayt kod + RET).

Adım 1 / 6

DEĞİŞKENLER

AL = 00h

BX = 0000h

VAR1 = 22h

Kod İzleme: LEA ile Değişken Güncelleme

1 **ORG** 100h

► **MOV** AL, VAR1 ; değer

3 **LEA** BX, VAR1 ; adres

4 **MOV** **BYTE PTR** [BX], 44h

5 **MOV** AL, VAR1 ; yeni değer

6 **RET**

7 VAR1 **DB** 22h

8 **END**

AX

00	22
AH	AL

BX

00	00
BH	BL

DS:010C ... (VAR1 = 010Dh)

010C	010D	010E
C3	22	00

AL = VAR1'in **değeri** = 22h.

Adım 2 / 6

DEĞİŞKENLER

AL = 22h

BX = 0000h

VAR1 = 22h

Kod İzleme: LEA ile Değişken Güncelleme

```
1 ORG 100h
2 MOV AL, VAR1          ; değer
3 LEA BX, VAR1          ; adres
4 MOV BYTE PTR [BX], 44h
5 MOV AL, VAR1          ; yeni değer
6 RET
7 VAR1 DB 22h
8 END
```

AX	00	22	BX	01	0D
	AH	AL		BH	BL

DS:010C ... (VAR1 = 010Dh)

010C	010D	010E
C3	22	00

BX = VAR1'in **adresi** = 010Dh. BX artık VAR1'i gösteren bir işaretçidir.

Adım 3 / 6

DEĞİŞKENLER

AL = 22h

BX = 010Dh

VAR1 = 22h

Kod İzleme: LEA ile Değişken Güncelleme

```
1 ORG 100h
2 MOV AL, VAR1          ; değer
3 LEA BX, VAR1          ; adres
4 ► MOV BYTE PTR [BX], 44h
5 MOV AL, VAR1          ; yeni değer
6 RET
7 VAR1 DB 22h
8 END
```

AX	00	22	BX	01	0D
	AH	AL		BH	BL

DS:010C ... (VAR1 = 010Dh)

010C	010D	010E
------	------	------

C3	44	00
----	----	----

[BX] üzerinden VAR1 güncellenir: [010Dh] = **44h**.

Adım 4 / 6

DEĞİŞKENLER

AL = 22h

BX = 010Dh

VAR1 = 44h

Kod İzleme: LEA ile Değişken Güncelleme

```
1 ORG 100h
2 MOV AL, VAR1           ; değer
3 LEA BX, VAR1           ; adres
4 MOV BYTE PTR [BX], 44h
5 ► MOV AL, VAR1         ; yeni değer
6 RET
7 VAR1 DB 22h
8 END
```

AX	00	44	BX	01	0D
	AH	AL		BH	BL

DS:010C ... (VAR1 = 010Dh)

010C	010D	010E
C3	44	00

VAR1 tekrar okunur: AL = **44h**. Değişkene adı ile de işaretçi ile de erişilebilir.

Adım 5 / 6

DEĞİŞKENLER

AL = 44h

BX = 010Dh

VAR1 = 44h

Kod İzleme: LEA ile Değişken Güncelleme

```
1 ORG 100h
2 MOV AL, VAR1           ; değer
3 LEA BX, VAR1           ; adres
4 MOV BYTE PTR [BX], 44h
5 MOV AL, VAR1           ; yeni değer
6 RET
7 VAR1 DB 22h
8 END
```

AX	00	44	BX	01	0D
	AH	AL		BH	BL

DS:010C ... (VAR1 = 010Dh)

010C	010D	010E
C3	44	00

RET. **END** kaynak dosyanın sonunu bildiren direktiftir.

Adım 6 / 6

DEĞİŞKENLER

AL = 44h

BX = 010Dh

VAR1 = 44h

LEA ve OFFSET Aynı mı?

OFFSET ile aynı program

```
ORG 100h
MOV AL, VAR1
MOV BX, OFFSET VAR1
MOV BYTE PTR [BX], 44h
MOV AL, VAR1
RET
VAR1 DB 22h
END
```

- emu8086'da `LEA BX, VAR1` ve `MOV BX, OFFSET VAR1` **aynı makine koduna** derlenir:
`MOV BX, num ( BB 0D 01 );` num, değişkenin bağlı konumunun 16 bitlik değeridir.
- MASM/TASM `LEA BX, VAR1` 'i gerçek LEA komutu olarak kodlar (`8D 1E 0D 01`, 4 bayt); sonuç yine aynıdır.
- Fark: `LEA BX, a[SI]` adresi **çalışma anında** hesaplar ($a + SI$); OFFSET yalnızca sabit adres verir.
- Köşeli parantez içinde yalnızca **BX, SI, DI, BP** kullanılabilir.

Kod İzleme: LEA a[BX] ile OFFSET a

```
1 ORG 100h
2 MOV BX, 2
3 LEA SI, a[BX]      ; SI = a + BX
4 MOV DI, OFFSET a  ; sabit adres
5 ADD DI, BX          ; elle ekle
6 MOV AL, [SI]
7 INC BX
8 LEA SI, a[BX]      ; yeni adres
9 MOV AH, [SI]
10 RET
11 a DB 'ABCD'
```

a



BX = 2: indis. a dizisi 0116h'de başlıyor: 'A' 'B' 'C' 'D'.

Adım 1 / 8

DEĞİŞKENLER

BX = 0002h

SI = 0000h

DI = 0000h

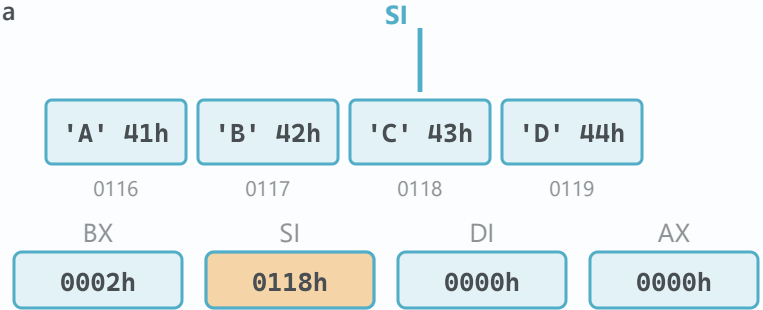
AX = 0000h

LEA adresi yazmaçlarla çalışma anında hesaplar; OFFSET yalnızca derleme anında bilinen sabit adresi verir.

Kod İzleme: LEA a[BX] ile OFFSET a

```
1 ORG 100h
2 MOV BX, 2
3 LEA SI, a[BX] ; SI = a + BX
4 MOV DI, OFFSET a ; sabit adres
5 ADD DI, BX ; elle ekle
6 MOV AL, [SI]
7 INC BX
8 LEA SI, a[BX] ; yeni adres
9 MOV AH, [SI]
10 RET
11 a DB 'ABCD'
```

a



LEA SI, a[BX] : adres **çalışma anında** hesaplanır: 0116h + 2 = **0118h**. Bellek okunmaz.

Adım 2 / 8

DEĞİŞKENLER

BX = 0002h

SI = 0118h

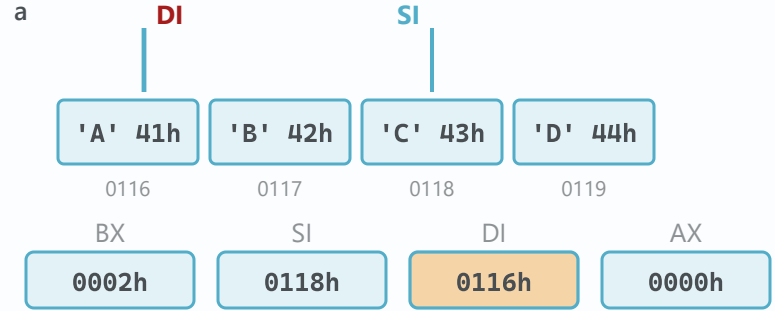
DI = 0000h

AX = 0000h

LEA adresi yazmaçlarla çalışma anında hesaplar; OFFSET yalnızca derleme anında bilinen sabit adresi verir.

Kod İzleme: LEA a[BX] ile OFFSET a

```
1 ORG 100h
2 MOV BX, 2
3 LEA SI, a[BX]      ; SI = a + BX
4 MOV DI, OFFSET a ; sabit adres
5 ADD DI, BX         ; elle ekle
6 MOV AL, [SI]
7 INC BX
8 LEA SI, a[BX]      ; yeni adres
9 MOV AH, [SI]
10 RET
11 a DB 'ABCD'
```



OFFSET a derleme anında bilinen sabittir: DI = **0116h**. BX'i katamaz.

Adım 3 / 8

DEĞİŞKENLER

BX = 0002h

SI = 0118h

DI = 0116h

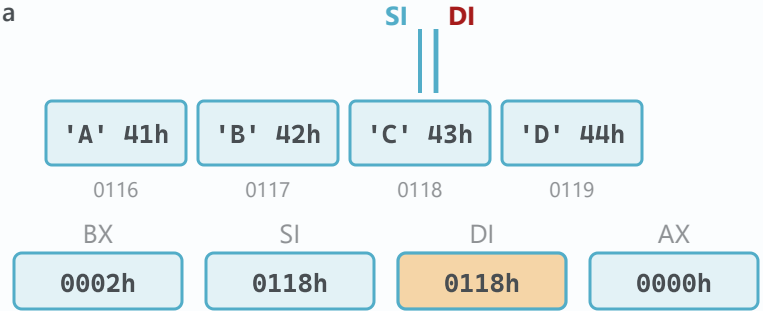
AX = 0000h

LEA adresi yazmaçlarla çalışma anında hesaplar; OFFSET yalnızca derleme anında bilinen sabit adresi verir.

Kod İzleme: LEA a[BX] ile OFFSET a

```
1 ORG 100h
2 MOV BX, 2
3 LEA SI, a[BX]      ; SI = a + BX
4 MOV DI, OFFSET a ; sabit adres
5 ADD DI, BX         ; elle ekle
6 MOV AL, [SI]
7 INC BX
8 LEA SI, a[BX]      ; yeni adres
9 MOV AH, [SI]
10 RET
11 a DB 'ABCD'
```

a



Aynı adres için OFFSET ile ayrıca **ADD** gerekir: DI = **0118h** = SI.

Adım 4 / 8

DEĞİŞKENLER

BX = 0002h

SI = 0118h

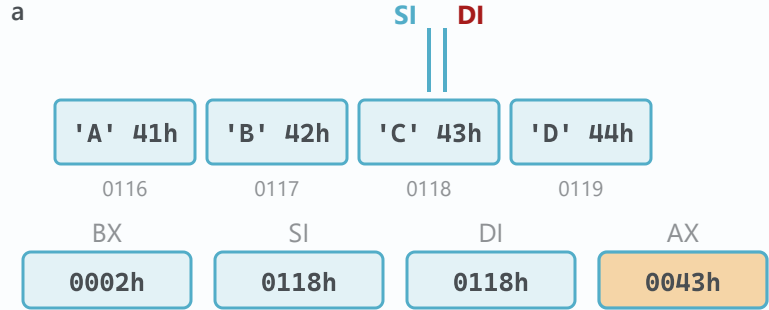
DI = 0118h

AX = 0000h

LEA adresi yazmaçlarla çalışma anında hesaplar; OFFSET yalnızca derleme anında bilinen sabit adresi verir.

Kod İzleme: LEA a[BX] ile OFFSET a

```
1 ORG 100h
2 MOV BX, 2
3 LEA SI, a[BX]      ; SI = a + BX
4 MOV DI, OFFSET a  ; sabit adres
5 ADD DI, BX          ; elle ekle
6 ► MOV AL, [SI]
7 INC BX
8 LEA SI, a[BX]      ; yeni adres
9 MOV AH, [SI]
10 RET
11 a DB 'ABCD'
```



AL = [SI] = 43h ('C').

Adım 5 / 8

DEĞİŞKENLER

BX = 0002h

SI = 0118h

DI = 0118h

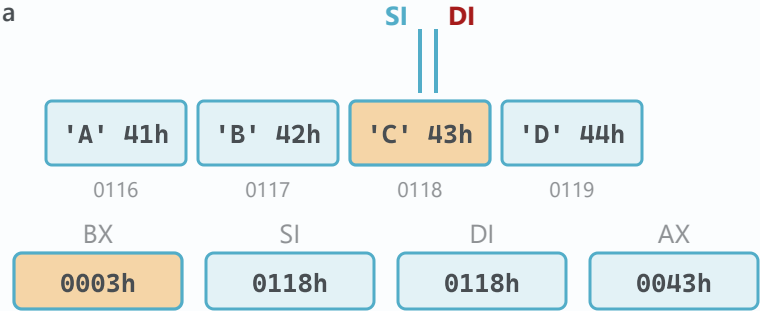
AX = 0043h

LEA adresi yazmaçlarla çalışma anında hesaplar; OFFSET yalnızca derleme anında bilinen sabit adresi verir.

Kod İzleme: LEA a[BX] ile OFFSET a

```
1 ORG 100h
2 MOV BX, 2
3 LEA SI, a[BX]      ; SI = a + BX
4 MOV DI, OFFSET a  ; sabit adres
5 ADD DI, BX          ; elle ekle
6 MOV AL, [SI]
7 INC BX
8 LEA SI, a[BX]      ; yeni adres
9 MOV AH, [SI]
10 RET
11 a DB 'ABCD'
```

a



BX = 3.

Adım 6 / 8

DEĞİŞKENLER

BX = 0003h

SI = 0118h

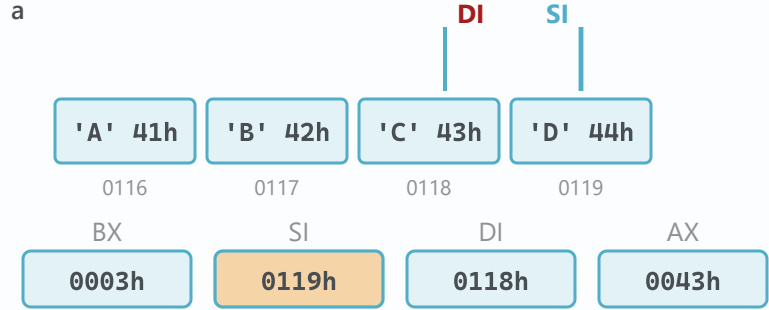
DI = 0118h

AX = 0043h

LEA adresi yazmaçlarla çalışma anında hesaplar; OFFSET yalnızca derleme anında bilinen sabit adresi verir.

Kod İzleme: LEA a[BX] ile OFFSET a

```
1 ORG 100h
2 MOV BX, 2
3 LEA SI, a[BX]      ; SI = a + BX
4 MOV DI, OFFSET a  ; sabit adres
5 ADD DI, BX          ; elle ekle
6 MOV AL, [SI]
7 INC BX
8 ► LEA SI, a[BX]     ; yeni adres
9 MOV AH, [SI]
10 RET
11 a DB 'ABCD'
```



LEA güncel BX ile yeniden hesaplar: SI = **0119h**.

Adım 7 / 8

DEĞİŞKENLER

BX = 0003h

SI = 0119h

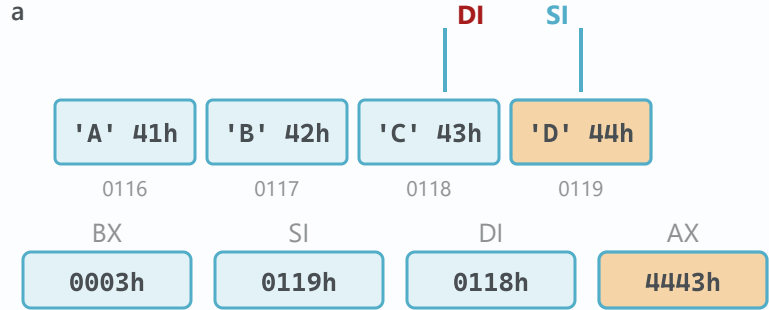
DI = 0118h

AX = 0043h

LEA adresi yazmaçlarla çalışma anında hesaplar; OFFSET yalnızca derleme anında bilinen sabit adresi verir.

Kod İzleme: LEA a[BX] ile OFFSET a

```
1 ORG 100h
2 MOV BX, 2
3 LEA SI, a[BX]      ; SI = a + BX
4 MOV DI, OFFSET a  ; sabit adres
5 ADD DI, BX          ; elle ekle
6 MOV AL, [SI]
7 INC BX
8 LEA SI, a[BX]      ; yeni adres
9 MOV AH, [SI]
10 RET
11 a DB 'ABCD'
```



AH = [SI] = **44h** ('D'). AX = 4443h.

Adım 8 / 8

DEĞİŞKENLER

BX = 0003h

SI = 0119h

DI = 0118h

AX = 4443h

LEA adresi yazmaçlarla çalışma anında hesaplar; OFFSET yalnızca derleme anında bilinen sabit adresi verir.

SABİTLER

Sabitler (EQU)

- Değişkenlere benzer, ancak yalnızca **derleme sırasında** var olurlar; bellekte yer kaplamazlar.
- Bir sabitin değeri tanımlandıktan sonra **değiştirilemez**.
- Programın farklı bölümlerinde aynı değerin kullanılmasını sağlar; değer tek yerden güncellenir.
- Tanım: `ad EQU <herhangi bir ifade>`
- Assembler her kullanımda adı değeriyle değiştirir:
`MOV AX, k` → `MOV AX, 5` (`B8 05 00`).

```
k      EQU 5
SATIR  EQU 80

MOV AX, k      ; AX = 5
MOV BX, SATIR*2 ; BX = 160
MOV CL, k+1    ; CL = 6
```

	k EQU 5	k DB 5
Bellek	kullanmaz	1 bayt
<code>MOV AL, k</code>	AL ← 5 (anlık)	AL ← [k] (bellekten)
Değiştirilebilir mi?	hayır	evet: <code>MOV k, 7</code>

Kod İzleme: EQU Komuta Gömülür

```

▶ k EQU 5
2 SATIR EQU 80
3 ORG 100h
4 MOV AX, k ; MOV AX, 5
5 MOV BX, SATIR*2 ; MOV BX, 160
6 MOV CL, k+1 ; MOV CL, 6
7 MOV DL, v ; bellekten
8 RET
9 v DB 5 ; 1 bayt yer

```

semboller (derleme)

k	5
AX	BX
0000h	0000h
CL	DL
00h	00h

bellek (kesim 0700h)

0100	B8 05 00	MOV AX, 5
0103	BB A0 00	MOV BX, 160
0106	B1 06	MOV CL, 6
0108	8A 16 0D 01	MOV DL, [010Dh]
010C	C3	RET
010D	05	v

k EQU 5 : assembler sembol tablosuna **k = 5** yazar. Bellekte yer ayrılmaz, makine kodu üretilmez.

Adım 1 / 6

DEĞİŞKENLER

AX = 0000h

BX = 0000h

CL = 00h

DL = 00h

Makine kodları MASM/emu8086 kodlamasıdır: B8 iw (MOV AX, sabit), BB iw (MOV BX, sabit), B1 ib (MOV CL, sabit), 8A 16 adres (MOV DL, [adres]).

Kod İzleme: EQU Komuta Gömülür

```

1 k EQU 5
▶ SATIR EQU 80
3 ORG 100h
4 MOV AX, k ; MOV AX, 5
5 MOV BX, SATIR*2 ; MOV BX, 160
6 MOV CL, k+1 ; MOV CL, 6
7 MOV DL, v ; bellekten
8 RET
9 v DB 5 ; 1 bayt yer

```

semboller (derleme)

k	5
SATIR	80
AX	
0000h	0000h
BX	
CL	
00h	00h
DL	

bellek (kesim 0700h)

0100	B8 05 00	MOV AX, 5
0103	BB A0 00	MOV BX, 160
0106	B1 06	MOV CL, 6
0108	8A 16 0D 01	MOV DL, [010Dh]
010C	C3	RET
010D	05	v

SATIR = 80 da yalnız derleme anında vardır. Program 0100h'den başlar; ilk bayt MOV AX, k 'nın kodudur.

Adım 2 / 6

DEĞİŞKENLER

AX = 0000h

BX = 0000h

CL = 00h

DL = 00h

Makine kodları MASM/emu8086 kodlamasıdır: B8 iw (MOV AX, sabit), BB iw (MOV BX, sabit), B1 ib (MOV CL, sabit), 8A 16 adres (MOV DL, [adres]).

Kod İzleme: EQU Komuta Gömülür

```

1 k      EQU 5
2 SATIR  EQU 80
3 ORG 100h
▶ MOV AX, k      ; MOV AX, 5
5 MOV BX, SATIR*2 ; MOV BX, 160
6 MOV CL, k+1     ; MOV CL, 6
7 MOV DL, v       ; bellekten
8 RET
9 v DB 5          ; 1 bayt yer

```

semboller (derleme)

k	5
SATIR	80
v	010Dh (adres)
AX	
0005h	0000h
CL	DL
00h	00h

bellek (kesim 0700h)

0100	B8 05 00	MOV AX, 5
0103	BB A0 00	MOV BX, 160
0106	B1 06	MOV CL, 6
0108	8A 16 0D 01	MOV DL, [010Dh]
010C	C3	RET
010D	05	v

MOV AX, k → MOV AX, 5 (B8 05 00): 5 komutun içinde,
anlık değer. AX = **0005h**.

Adım 3 / 6

DEĞİŞKENLER

AX = 0005h

BX = 0000h

CL = 00h

DL = 00h

Makine kodları MASM/emu8086 kodlamasıdır: B8 iw (MOV AX, sabit), BB iw (MOV BX, sabit), B1 ib (MOV CL, sabit), 8A 16 adres (MOV DL, [adres]).

Kod İzleme: EQU Komuta Gömülür

```

1 k      EQU 5
2 SATIR  EQU 80
3 ORG 100h
4 MOV AX, k      ; MOV AX, 5
5 ► MOV BX, SATIR*2 ; MOV BX, 160
6 MOV CL, k+1    ; MOV CL, 6
7 MOV DL, v      ; bellekten
8 RET
9 v DB 5        ; 1 bayt yer

```

semboller (derleme)

k	5
SATIR	80
v	010Dh (adres)
AX	
0005h	
CL	
00h	
BX	
00A0h	
DL	
00h	

bellek (kesim 0700h)

0100	B8 05 00	MOV AX, 5
0103	BB A0 00	MOV BX, 160
0106	B1 06	MOV CL, 6
0108	8A 16 0D 01	MOV DL, [010Dh]
010C	C3	RET
010D	05	v

SATIR*2 derleme anında hesaplandı: 160 = 00A0h → **BX = 00A0h**.

Adım 4 / 6

DEĞİŞKENLER

AX = 0005h

BX = 00A0h

CL = 00h

DL = 00h

Makine kodları MASM/emu8086 kodlamasıdır: B8 iw (MOV AX, sabit), BB iw (MOV BX, sabit), B1 ib (MOV CL, sabit), 8A 16 adres (MOV DL, [adres]).

Kod İzleme: EQU Komuta Gömülür

```

1 k      EQU 5
2 SATIR  EQU 80
3 ORG 100h
4 MOV AX, k      ; MOV AX, 5
5 MOV BX, SATIR*2 ; MOV BX, 160
6 ► MOV CL, k+1   ; MOV CL, 6
7 MOV DL, v      ; bellekten
8 RET
9 v DB 5         ; 1 bayt yer

```

semboller (derleme)

k	5
SATIR	80
v	010Dh (adres)
AX	
0005h	00A0h
CL	DL
06h	00h

bellek (kesim 0700h)

0100	B8 05 00	MOV AX, 5
0103	BB A0 00	MOV BX, 160
0106	B1 06	MOV CL, 6
0108	8A 16 0D 01	MOV DL, [010Dh]
010C	C3	RET
010D	05	v

$k+1 = 6 \rightarrow CL = 06h$. Çalışma anında toplama yapılmadı.

Adım 5 / 6

DEĞİŞKENLER

AX = 0005h

BX = 00A0h

CL = 06h

DL = 00h

Makine kodları MASM/emu8086 kodlamasıdır: B8 iw (MOV AX, sabit), BB iw (MOV BX, sabit), B1 ib (MOV CL, sabit), 8A 16 adres (MOV DL, [adres]).

Kod İzleme: EQU Komuta Gömülür

```

1 k      EQU 5
2 SATIR  EQU 80
3 ORG 100h
4 MOV AX, k      ; MOV AX, 5
5 MOV BX, SATIR*2 ; MOV BX, 160
6 MOV CL, k+1    ; MOV CL, 6
7 ► MOV DL, v    ; bellekten
8 RET
9 v DB 5        ; 1 bayt yer

```

semboller (derleme)

k	5
SATIR	80
v	010Dh (adres)
AX	
0005h	00A0h
CL	DL
06h	05h

bellek (kesim 0700h)

0100	B8 05 00	MOV AX, 5
0103	BB A0 00	MOV BX, 160
0106	B1 06	MOV CL, 6
0108	8A 16 0D 01	MOV DL, [010Dh]
010C	C3	RET
010D	05	v

v bir değişken: komut adresini (010Dh) taşır, değer **bellekten** okunur: DL = 05h. v 1 bayt yer kaplar, k hiç kaplamaz.

Adım 6 / 6

DEĞİŞKENLER

AX = 0005h

BX = 00A0h

CL = 06h

DL = 05h

Makine kodları MASM/emu8086 kodlamasıdır: B8 iw (MOV AX, sabit), BB iw (MOV BX, sabit), B1 ib (MOV CL, sabit), 8A 16 adres (MOV DL, [adres]).

Kod İzleme: Değişkenler Arasında Veri Taşıma

► **jmp** start

```
2 value1 DW 10      ; 2 baytlık
   değişken
3 value2 DW 0        ; ikinci değişken
4 start:
5 MOV AX, value1     ; AX ← value1
6 MOV value2, AX     ; value2 ← AX
7 end start
```

AX 00 00
 AH AL

değişkenler (DS)

value1	value1+1	value2	value2+1
0A	00	00	00

jmp start CPU'yu verilerin **üzerinden atlatır**; aksi hâlde veri baytları komut gibi yürütülürdü.

Adım 1 / 4

DEĞİŞKENLER

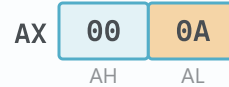
AX = 0000h

value1 = 000Ah

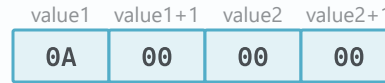
value2 = 0000h

Kod İzleme: Değişkenler Arasında Veri Taşıma

```
1 jmp start
2 value1 DW 10      ; 2 baytlık
   değişken
3 value2 DW 0        ; ikinci değişken
4 start:
▶ MOV AX, value1     ; AX ← value1
6 MOV value2, AX     ; value2 ← AX
7 end start
```



değişkenler (DS)



value1 → AX = **000Ah**. Bellekten belleğe doğrudan MOV olmadığı için ara yazmaç gerekir.

Adım 2 / 4

DEĞİŞKENLER

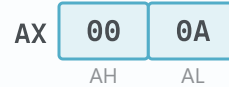
AX = **000Ah**

value1 = **000Ah**

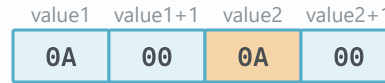
value2 = **0000h**

Kod İzleme: Değişkenler Arasında Veri Taşıma

```
1 jmp start
2 value1 DW 10      ; 2 baytlık
   değişken
3 value2 DW 0       ; ikinci değişken
4 start:
5 MOV AX, value1    ; AX ← value1
▶ MOV value2, AX    ; value2 ← AX
7 end start
```



değişkenler (DS)



AX → value2: bellekte **0A 00**. Artık value1 = value2 = 10.

Adım 3 / 4

DEĞİŞKENLER

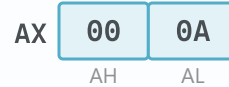
AX = 000Ah

value1 = 000Ah

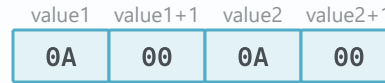
value2 = 000Ah

Kod İzleme: Değişkenler Arasında Veri Taşıma

```
1 jmp start
2 value1 DW 10      ; 2 baytlık
   değişken
3 value2 DW 0        ; ikinci değişken
4 start:
5 MOV AX, value1     ; AX ← value1
6 MOV value2, AX     ; value2 ← AX
▶ end start
```



değişkenler (DS)



end start : kaynağın sonu; **start** programın giriş noktasıdır.

Adım 4 / 4

DEĞİŞKENLER

AX = 000Ah

value1 = 000Ah

value2 = 000Ah

Kod İzleme: Değişkenleri Toplama

► **jmp** start

```
2 value1 DW 105h ; ilk değer 105h
3 value2 DW 103h ; ilk değer 103h
4 result DW ? ; belirsiz
5 start:
6 MOV AX, value1 ; AX ← value1
7 ADD AX, value2 ; AX ← AX + value2
8 MOV result, AX ; result ← AX
9 end start
```

AX

00	00
AH	AL

Bayraklar

CF=0	ZF=0	SF=0	OF=0	PF=0	AF=0
------	------	------	------	------	------

değişkenler (DS)

value1	value1+1	value2	value2+1	result	result+1
05	01	03	01	00	00

Başlangıç: result

Adım 1 / 5

DEĞİŞKENLER

AX = 0000h

CF = 0

ZF = 0

SF = 0

OF = 0

PF = 0

AF = 0

value1 = 0105h

value2 = 0103h

result = 0000h

Kod İzleme: Değişkenleri Toplama

```
1 jmp start
2 value1 DW 105h ; ilk değer 105h
3 value2 DW 103h ; ilk değer 103h
4 result DW ? ; belirsiz
5 start:
▶ MOV AX, value1 ; AX ← value1
7 ADD AX, value2 ; AX ← AX + value2
8 MOV result, AX ; result ← AX
9 end start
```

AX

01	05
AH	AL

Bayraklar

CF=0	ZF=0	SF=0	OF=0	PF=0	AF=0
------	------	------	------	------	------

değişkenler (DS)

value1	value1+1	value2	value2+1	result	result+1
05	01	03	01	00	00

AX = value1 = **0105h**.

Adım 2 / 5

DEĞİŞKENLER

AX = **0105h**

CF = 0

ZF = 0

SF = 0

OF = 0

PF = 0

AF = 0

value1 = **0105h**

value2 = **0103h**

result = **0000h**

Kod İzleme: Değişkenleri Toplama

```
1 jmp start
2 value1 DW 105h ; ilk değer 105h
3 value2 DW 103h ; ilk değer 103h
4 result DW ? ; belirsiz
5 start:
6 MOV AX, value1 ; AX ← value1
7 ► ADD AX, value2 ; AX ← AX + value2
8 MOV result, AX ; result ← AX
9 end start
```

AX

02	08
AH	AL

Bayraklar

CF=0	ZF=0	SF=0	OF=0	PF=0	AF=0
------	------	------	------	------	------

değişkenler (DS)

value1	value1+1	value2	value2+1	result	result+1
05	01	03	01	00	00

ADD bellek işleneni doğrudan kullanabilir: 0105h + 0103h = **0208h**. Bayraklar: CF = 0, ZF = 0, PF = 0 (08h'de bir adet 1).

Adım 3 / 5

DEĞİŞKENLER

AX = 0208h

CF = 0

ZF = 0

SF = 0

OF = 0

PF = 0

AF = 0

value1 = 0105h

value2 = 0103h

result = 0000h

Kod İzleme: Değişkenleri Toplama

```

1 jmp start
2 value1 DW 105h ; ilk değer 105h
3 value2 DW 103h ; ilk değer 103h
4 result DW ? ; belirsiz
5 start:
6 MOV AX, value1 ; AX ← value1
7 ADD AX, value2 ; AX ← AX + value2
▶ MOV result, AX ; result ← AX
9 end start
    
```

AX

02	08
----	----

AH AL

Bayraklar

CF=0	ZF=0	SF=0	OF=0	PF=0	AF=0
------	------	------	------	------	------

değişkenler (DS)

value1	value1+1	value2	value2+1	result	result+1
05	01	03	01	08	02

result ← AX: bellekte **08 02** (little-endian).

Adım 4 / 5

DEĞİŞKENLER

AX = **0208h**

CF = 0

ZF = 0

SF = 0

OF = 0

PF = 0

AF = 0

value1 = **0105h**

value2 = **0103h**

result = **0208h**

Kod İzleme: Değişkenleri Toplama

```
1 jmp start
2 value1 DW 105h ; ilk değer 105h
3 value2 DW 103h ; ilk değer 103h
4 result DW ? ; belirsiz
5 start:
6 MOV AX, value1 ; AX ← value1
7 ADD AX, value2 ; AX ← AX + value2
8 MOV result, AX ; result ← AX
▶ end start
```

AX

02	08
AH	AL

Bayraklar

CF=0	ZF=0	SF=0	OF=0	PF=0	AF=0
------	------	------	------	------	------

değişkenler (DS)

value1	value1+1	value2	value2+1	result	result+1
05	01	03	01	08	02

Sonuç: result = 0208h = 520 (261 + 259).

Adım 5 / 5

DEĞİŞKENLER

AX = 0208h

CF = 0

ZF = 0

SF = 0

OF = 0

PF = 0

AF = 0

value1 = 0105h

value2 = 0103h

result = 0208h

Kod İzleme: Büyük Olanı Bul (10, 5)

```
1 ORG 100h
▶ 2 MOV AX, value1      ; AX ← value1
3 CMP AX, value2        ; AX - value2
4 JGE greater_or_equal
5 MOV AX, value2        ; küçükse value2
6 JMP store_max
7 greater_or_equal:
8 MOV AX, value1
9 store_max:
10 MOV max_value, AX    ; sonucu sakla
11 RET
12 value1 DW 10
13 value2 DW 5
14 max_value DW ?
```

AX

00	0A
AH	AL

Bayraklar

CF=0

ZF=0

SF=0

OF=0

değişkenler (DS)

value1	value1+1	value2	value2+1	max_value	max_value+1
0A	00	05	00	00	00

AX = value1 = **000Ah**.

Adım 1 / 6

DEĞİŞKENLER

AX = **000Ah**

CF = **0**

ZF = **0**

SF = **0**

OF = **0**

value1 = **000Ah**

value2 = **0005h**

max_value = **0000h**

Kod İzleme: Büyük Olanı Bul (10, 5)

```
1 ORG 100h
2 MOV AX, value1      ; AX ← value1
3 CMP AX, value2      ; AX - value2
4 JGE greater_or_equal
5 MOV AX, value2      ; küçükse value2
6 JMP store_max
7 greater_or_equal:
8 MOV AX, value1
9 store_max:
10 MOV max_value, AX  ; sonucu sakla
11 RET
12 value1 DW 10
13 value2 DW 5
14 max_value DW ?
```

AX 00 0A
 AH AL

Bayraklar

CF=0

ZF=0

SF=0

OF=0

değişkenler (DS)

value1	value1+1	value2	value2+1	max_value	max_value+1
0A	00	05	00	00	00

CMP çıkarma yapar ama sonucu **saklamaz**, yalnız bayrakları ayarlar: 10 – 5.

Adım 2 / 6

DEĞİŞKENLER

AX = 000Ah

CF = 0

ZF = 0

SF = 0

OF = 0

value1 = 000Ah

value2 = 0005h

max_value = 0000h

Kod İzleme: Büyük Olanı Bul (10, 5)

```

1  ORG 100h
2  MOV AX, value1      ; AX ← value1
3  CMP AX, value2      ; AX - value2
4  JGE greater_or_equal
5  MOV AX, value2      ; küçükse value2
6  JMP store_max
7  greater_or_equal:
8  MOV AX, value1
9  store_max:
10 MOV max_value, AX   ; sonucu sakla
11 RET
12 value1 DW 10
13 value2 DW 5
14 max_value DW ?
    
```

AX **00** **0A**
 AH AL

Bayraklar

CF=0 **ZF=0** **SF=0** **OF=0**

değişkenler (DS)

value1	value1+1	value2	value2+1	max_value	max_value+1
0A	00	05	00	00	00

JGE (işaretli $\geq$): SF = OF olduğundan **atlanır** → greater_or_equal.

Adım 3 / 6

DEĞİŞKENLER

AX = 000Ah	CF = 0	ZF = 0	SF = 0	OF = 0
value1 = 000Ah	value2 = 0005h	max_value = 0000h		

Kod İzleme: Büyük Olanı Bul (10, 5)

```
1 ORG 100h
2 MOV AX, value1      ; AX ← value1
3 CMP AX, value2      ; AX - value2
4 JGE greater_or_equal
5 MOV AX, value2      ; küçükse value2
6 JMP store_max
7 greater_or_equal:
8 MOV AX, value1
9 store_max:
10 MOV max_value, AX ; sonucu sakla
11 RET
12 value1 DW 10
13 value2 DW 5
14 max_value DW ?
```

AX

00	0A
AH	AL

Bayraklar

CF=0	ZF=0	SF=0	OF=0
------	------	------	------

değişkenler (DS)

value1	value1+1	value2	value2+1	max_value	max_value+1
0A	00	05	00	00	00

AX = value1 (zaten öyleydi; bu satır gereksiz ama zararsız).

Adım 4 / 6

DEĞİŞKENLER

AX = 000Ah

CF = 0

ZF = 0

SF = 0

OF = 0

value1 = 000Ah

value2 = 0005h

max_value = 0000h

Kod İzleme: Büyük Olanı Bul (10, 5)

```
1 ORG 100h
2 MOV AX, value1      ; AX ← value1
3 CMP AX, value2      ; AX - value2
4 JGE greater_or_equal
5 MOV AX, value2      ; küçükse value2
6 JMP store_max
7 greater_or_equal:
8 MOV AX, value1
9 store_max:
10 MOV max_value, AX ; sonucu sakla
11 RET
12 value1 DW 10
13 value2 DW 5
14 max_value DW ?
```

AX

00	0A
AH	AL

Bayraklar

CF=0	ZF=0	SF=0	OF=0
------	------	------	------

değişkenler (DS)

value1	value1+1	value2	value2+1	max_value	max_value+1
0A	00	05	00	0A	00

max_value ← AX = **000Ah**.

Adım 5 / 6

DEĞİŞKENLER

AX = **000Ah**

CF = 0

ZF = 0

SF = 0

OF = 0

value1 = 000Ah

value2 = 0005h

max_value = **000Ah**

Kod İzleme: Büyük Olanı Bul (10, 5)

```
1 ORG 100h
2 MOV AX, value1      ; AX ← value1
3 CMP AX, value2      ; AX - value2
4 JGE greater_or_equal
5 MOV AX, value2      ; küçükse value2
6 JMP store_max
7 greater_or_equal:
8 MOV AX, value1
9 store_max:
10 MOV max_value, AX ; sonucu sakla
11 RET
12 value1 DW 10
13 value2 DW 5
14 max_value DW ?
```

AX

00	0A
AH	AL

Bayraklar

CF=0

ZF=0

SF=0

OF=0

değişkenler (DS)

value1	value1+1	value2	value2+1	max_value	max_value+1
0A	00	05	00	0A	00

RET: DOS'a dönüş. RET olmasaydı CPU value1'in baytlarını komut sanıp yürütürdü.

Adım 6 / 6

DEĞİŞKENLER

AX = 000Ah

CF = 0

ZF = 0

SF = 0

OF = 0

value1 = 000Ah

value2 = 0005h

max_value = 000Ah

Kod İzleme: Büyük Olanı Bul (3, 7)

```
1 ORG 100h
▶ 2 MOV AX, value1      ; AX ← value1
3 CMP AX, value2        ; AX - value2
4 JGE greater_or_equal
5 MOV AX, value2        ; küçükse value2
6 JMP store_max
7 greater_or_equal:
8 MOV AX, value1
9 store_max:
10 MOV max_value, AX    ; sonucu sakla
11 RET
12 value1 DW 10
13 value2 DW 5
14 max_value DW ?
```

AX

00	03
AH	AL

Bayraklar

CF=0	ZF=0	SF=0	OF=0
------	------	------	------

değişkenler (DS)

value1	value1+1	value2	value2+1	max_value	max_value+1
03	00	07	00	00	00

AX = value1 = **0003h**.

Adım 1 / 7

DEĞİŞKENLER

AX = **0003h**

CF = 0

ZF = 0

SF = 0

OF = 0

value1 = **0003h**

value2 = **0007h**

max_value = **0000h**

Kod İzleme: Büyük Olanı Bul (3, 7)

```
1 ORG 100h
2 MOV AX, value1      ; AX ← value1
3 ► CMP AX, value2     ; AX - value2
4 JGE greater_or_equal
5 MOV AX, value2      ; küçükse value2
6 JMP store_max
7 greater_or_equal:
8 MOV AX, value1
9 store_max:
10 MOV max_value, AX  ; sonucu sakla
11 RET
12 value1 DW 10
13 value2 DW 5
14 max_value DW ?
```

AX 00 03
 AH AL

Bayraklar

CF=1

ZF=0

SF=1

OF=0

değişkenler (DS)

value1	value1+1	value2	value2+1	max_value	max_value+1
03	00	07	00	00	00

CMP çıkarma yapar ama sonucu **saklamaz**, yalnız bayrakları ayarlar: 3 – 7.

Adım 2 / 7

DEĞİŞKENLER

AX = 0003h

CF = 1

ZF = 0

SF = 1

OF = 0

value1 = 0003h

value2 = 0007h

max_value = 0000h

Kod İzleme: Büyük Olanı Bul (3, 7)

```
1 ORG 100h
2 MOV AX, value1      ; AX ← value1
3 CMP AX, value2      ; AX - value2
4 JGE greater_or_equal
5 MOV AX, value2      ; küçükse value2
6 JMP store_max
7 greater_or_equal:
8 MOV AX, value1
9 store_max:
10 MOV max_value, AX  ; sonucu sakla
11 RET
12 value1 DW 10
13 value2 DW 5
14 max_value DW ?
```

AX

00	03
AH	AL

Bayraklar

CF=1

ZF=0

SF=1

OF=0

değişkenler (DS)

value1	value1+1	value2	value2+1	max_value	max_value+1
03	00	07	00	00	00

SF ≠ OF → koşul sağlanmaz, **atlanmaz**; sıradaki komuta geçilir.

Adım 3 / 7

DEĞİŞKENLER

AX = 0003h

CF = 1

ZF = 0

SF = 1

OF = 0

value1 = 0003h

value2 = 0007h

max_value = 0000h

Kod İzleme: Büyük Olanı Bul (3, 7)

```
1 ORG 100h
2 MOV AX, value1      ; AX ← value1
3 CMP AX, value2      ; AX - value2
4 JGE greater_or_equal
▶ MOV AX, value2      ; küçükse value2
6 JMP store_max
7 greater_or_equal:
8 MOV AX, value1
9 store_max:
10 MOV max_value, AX ; sonucu sakla
11 RET
12 value1 DW 10
13 value2 DW 5
14 max_value DW ?
```

AX

00	07
AH	AL

Bayraklar

CF=1

ZF=0

SF=1

OF=0

değişkenler (DS)

value1	value1+1	value2	value2+1	max_value	max_value+1
03	00	07	00	00	00

AX = value2 = **0007h** (büyük olan).

Adım 4 / 7

DEĞİŞKENLER

AX = **0007h**

CF = 1

ZF = 0

SF = 1

OF = 0

value1 = **0003h**

value2 = **0007h**

max_value = **0000h**

Kod İzleme: Büyük Olanı Bul (3, 7)

```
1 ORG 100h
2 MOV AX, value1      ; AX ← value1
3 CMP AX, value2      ; AX - value2
4 JGE greater_or_equal
5 MOV AX, value2      ; küçükse value2
6 JMP store_max
7 greater_or_equal:
8 MOV AX, value1
9 store_max:
10 MOV max_value, AX  ; sonucu sakla
11 RET
12 value1 DW 10
13 value2 DW 5
14 max_value DW ?
```

AX

00	07
AH	AL

Bayraklar

CF=1

ZF=0

SF=1

OF=0

değişkenler (DS)

value1	value1+1	value2	value2+1	max_value	max_value+1
03	00	07	00	00	00

JMP koşulsuz atlar → store_max.

Adım 5 / 7

DEĞİŞKENLER

AX = 0007h

CF = 1

ZF = 0

SF = 1

OF = 0

value1 = 0003h

value2 = 0007h

max_value = 0000h

Kod İzleme: Büyük Olanı Bul (3, 7)

```
1 ORG 100h
2 MOV AX, value1      ; AX ← value1
3 CMP AX, value2      ; AX - value2
4 JGE greater_or_equal
5 MOV AX, value2      ; küçükse value2
6 JMP store_max
7 greater_or_equal:
8 MOV AX, value1
9 store_max:
10 MOV max_value, AX ; sonucu sakla
11 RET
12 value1 DW 10
13 value2 DW 5
14 max_value DW ?
```

AX

00	07
AH	AL

Bayraklar

CF=1

ZF=0

SF=1

OF=0

değişkenler (DS)

value1	value1+1	value2	value2+1	max_value	max_value+1
03	00	07	00	07	00

max_value ← AX = **0007h**.

Adım 6 / 7

DEĞİŞKENLER

AX = **0007h**

CF = 1

ZF = 0

SF = 1

OF = 0

value1 = 0003h

value2 = 0007h

max_value = **0007h**

Kod İzleme: Büyük Olanı Bul (3, 7)

```
1 ORG 100h
2 MOV AX, value1      ; AX ← value1
3 CMP AX, value2      ; AX - value2
4 JGE greater_or_equal
5 MOV AX, value2      ; küçükse value2
6 JMP store_max
7 greater_or_equal:
8 MOV AX, value1
9 store_max:
10 MOV max_value, AX ; sonucu sakla
11 RET
12 value1 DW 10
13 value2 DW 5
14 max_value DW ?
```

AX

00	07
AH	AL

Bayraklar

CF=1

ZF=0

SF=1

OF=0

değişkenler (DS)

value1	value1+1	value2	value2+1	max_value	max_value+1
03	00	07	00	07	00

RET: DOS'a dönüş. RET olmasaydı CPU value1'in baytlarını komut sanıp yürütürdü.

Adım 7 / 7

DEĞİŞKENLER

AX = 0007h

CF = 1

ZF = 0

SF = 1

OF = 0

value1 = 0003h

value2 = 0007h

max_value = 0007h

Karşılaştırma Notları

CMP

`CMP a, b` = `a - b` hesaplanır, sonuç atılır; yalnız bayraklar (CF, ZF, SF, OF, PF, AF) güncellenir.

İşaretili / işaretsiz

`JGE`, `JG`, `JL`: işaretili (SF, OF). `JAE`, `JA`, `JB`: işaretsiz (CF). `value1` = -1 (FFFFh), `value2` = 5 için `JGE` atlamaz, `JAE` atlar.

Neden RET?

Veri tanımları kodun hemen ardından gelir; `RET` olmasaydı CPU `0A 00 05 00...` baytlarını komut olarak yürütürdü. Çözüm: `RET` (ya da başta `jmp start`).

İyileştirme

`greater_or_equal` altındaki `MOV AX, value1` gereksizdir; AX zaten `value1`'dir. Etiket doğrudan `store_max`'e atlayabilir.

JAE (işaretsiz)

0 5

FFFFh ≥ 0005h ? işaretsiz evet, işaretili hayır

65535 (FFFFh)

JGE (işaretili)

-32768

-1 0 5

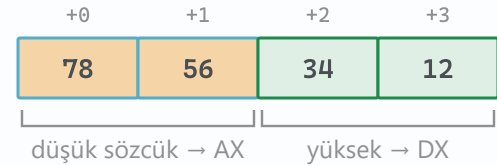
32767

ÖRNEK PROGRAMLAR

Çift Sözcük Değişken

```
jmp start
mydouble dd 12345678h ; double word
mywords dw 5678h ; 2 ayrı word
          dw 1234h
mybytes db 78h ; 4 ayrı byte
          db 56h
          db 34h
          db 12h
data dd 00010010001101000101011001111000b
start: ; data'yı DX:AX'e yükle
mov ax, word ptr data
mov dx, word ptr [data+2]
```

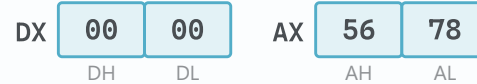
Dört tanımın da bellekteki karşılığı aynıdır:



- **DD** 32 bit = 4 bayt; en düşük bayt en düşük adreste.
- 8086'nın 32 bitlik yazmacı yoktur: değer **DX:AX** çiftinde tutulur (DX yüksek, AX düşük sözcük).

Kod İzleme: DX:AX'e 32 Bitlik Yükleme

```
► mov ax, word ptr data ; data+0  
2 mov dx, word ptr [data+2] ; data+2  
3 data dd 12345678h
```



data (4 bayt)



data'nın ilk iki baytı (78 56) → AX = **5678h**. `word ptr` :
data DD olduğu için boyut açıkça belirtilir.

Adım 1 / 2

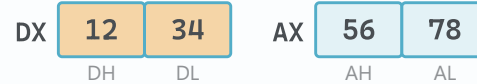
DEĞİŞKENLER

DX = 0000h

AX = 5678h

Kod İzleme: DX:AX'e 32 Bitlik Yükleme

```
1 mov ax, word ptr data ; data+0  
▶ mov dx, word ptr [data+2] ; data+2  
3 data dd 12345678h
```



data (4 bayt)



data+2'deki iki bayt (34 12) → DX = **1234h**. DX:AX = **1234:5678h** = 12345678h.

Adım 2 / 2

DEĞİŞKENLER

DX = 1234h

AX = 5678h

ÖRNEK PROGRAMLAR

Veri Tanımlama Özeti

Yazım	Anlamı	Bellek
x DB 7	bir bayt	07
y DW 1234h	bir sözcük	34 12
z DD 12345678h	çift sözcük	78 56 34 12
s DB 'Hi', 0	dizgi + sonlandırıcı	48 69 00
t DB 3 DUP(0)	tekrar	00 00 00
u DW ?	ilk değersiz	?? ?? (emu8086: 00 00)
k EQU 10	sabit	yer kaplamaz

Adres: LEA BX, x / MOV BX, OFFSET x . Değer: MOV AL, x . Boyut değiştirme: word ptr , byte ptr .

ÖZET

Özet

Değişken

Bir bellek konumuna verilen ad; derleyici adı bağlı konumla değiştirir.

DB / DW / DD

1, 2, 4 bayt; little-endian; ? ilk değersiz.

ORG 100h

.COM kodu 0100h'den başlar; ilk 256 bayt PSP. Veriler RET'ten sonra ya da jmp ile atlanarak.

Diziler

Ardışık baytlar; `a[i]` bayt uzaklığıdır; DUP ile tekrar; DW dizisinde indis $2 \times i$.

LEA / OFFSET

Adres almak için; LEA çalışma anında indisli adres de hesaplar.

EQU

Derleme anı sabiti; bellekte yer kaplamaz, değiştirilemez.

ÖZET

Alıştırmalar

1. `x DB 5, 6` · `y DW 0ABCDh` · `z DB 'OK'` tanımları 0200h'den başlıyorsa her baytın adresi ve değeri?
2. `tablo DW 10 DUP(?)` kaç bayt kaplar? 7. elemanın (indis 6) bağlı uzaklığı nedir?
3. Önceki sorudaki `y` için `MOV AL, BYTE PTR y` ve `MOV AL, BYTE PTR y+1` sonrası AL?
4. `LEA SI, a[BX]` ile `MOV SI, OFFSET a` arasındaki fark nedir?
5. İki DW değişkenin toplamını üçüncüsüne yazan, toplamda taşma olursa (CF = 1) `tasma DB 1` yapan kodu yazın.
6. `dd 0FFFFFFFFh` değerine 1 eklemek için DX:AX'te hangi komut çifti gerekir?

► Cevaplar

Kaynaklar

- emu8086 belgeleri, *8086 assembler tutorial for beginners*, bölüm 3: variables (DB, DW, DUP, LEA, OFFSET, EQU).
- Barry B. Brey, *The Intel Microprocessors*, 8. baskı, Pearson.
- Kip R. Irvine, *Assembly Language for x86 Processors*, Pearson (veri tanımlama direktifleri).
- Microsoft MASM 6.1 Programmer's Guide: DB/DW/DD, DUP, EQU, OFFSET.

Son

Bölüm 4: Değişkenler

Sonraki bölüm: Bölüm 5: Kesmeler · [Terim sözlüğü](#)